FSEFI Installation and User Manual

Qiang Guan, Ph.D.

Ultrascale Systems Research Center (USRC)

High Performance Computing (HPC) division

Los Alamos National Laboratory (LANL)

under contract by U.S. Department of Energy (DOE)

Feb 19th, 2015: Release 1.0 on Ubuntu 12.04 (based on TEMU-1.0)

Contents

1 Attribution

The Fine-grained Soft Error Fault Injector (FSEFI) is a software fault injector developped by Los Alamos National Laboratory (LANL) since 2011. LANL is managed by the U.S. Department of Energy (DOE). F-SEFI was developed at the Ultrascale Systems Research Center (USRC) at the New Mexico Consortium. Its creators and maintainers are from the High Performance Computing (HPC) division of LANL and has these contributors:

1. Nathan DeBardeleben, Ph.D. (ndebard@lanl.gov) - project PI, manager, etc.
2. Qiang Guan, Ph.D. (qguan@lanl.gov) - primary developer, maintainer, research
3. Sean Blanchard (seanb@lanl.gov) - operating systems, kernels, etc.
4. Conor Robinson - F-SEFI usage, experiments, etc.
5. Laura Monroe, Ph.D. - algorithms
6. William Jones, Ph.D. - algorithms, Coastal Carolina University
7. Claude "Rusty" Davis - algorithms, experiments, Clemson University

All questions and inquiries about F-SEFI should be directed to Nathan DeBardeleben at Los Alamos National Laboratory - ndebard@lanl.gov.

1 Introduction

F-SEFI capitalizes on extensive open source work in the QEMU/TEMU processor emulator virtal machine (VM). F-SEFI connects to the QEMU hypervisor and allows controlled fault injection into the VM from the host. All of the injection operations runs entirely in user space from the host aspect but from the VM guest aspect, where the applications runs, the injection operations runs in hardwares. This feature enables us to emulate the faults in hardware without any instrusions to kernel or privelges on both host and guest VMs. FSEFI supports two levels of fault injections: Course-grained and Fine-grained.

Course-grained
- Injecting faults into target application level. Faults are injected into the application at any time in any functions that associates with your target operation.

Fine-grained
- Injecting faults into the sub-function level of the target application. Faults are injected into the function even target lines-of-code (LOC) with the support of Function Symbol Table (FST)

This document is a quick start guide for setting up and running FSEFI. The instructions are based on the release of FSEFI-1.0 shown in the header, running on a vanilla Ubuntu 12.04 64-bits distribution of Linux . We intermix instructions with explanations about functions to provide a tutorial of how FSEFI works on injecting faults. The goal in this exercise is to take a simple program, profile the program, configure the fault injector based on the profile, execute the fault inejction at runtime, trace the injection via F-SEFI logs.

2 Installation

If you are a new user, please log in your account and cp all the files (inlcuding source code, scripts and image file) from `/home/fsefi_shared/` directory

```
$ cp -r /home/fsefi_shared/* ~/
$ ls
FSEFI_CURRENT image install-fsefi.sh
```

The following script shows the steps for building and installing FSEFI and the other software it depends on: (This is also found as `docs/install-fsefi.sh` in the code base, you can simply run it to install FSEFI and all the packages). We have another script for installing all the dependency packages that requre for building FSEFI.

```
#!/bin/bash
# Instructions for installing FSEFI 1.0 on Ubuntu 12.04 Linux 64-bit

# You will see the folder called FSEFI_CURRENT from extracting the tarball
cd FSEFI_CURRENT

# Build the sleuthkit-2.04
(cd shared/sleuthkit && make)
(cd shared && ln -s sleuthkit-2.04 sleuthkit)

# Build the llconf
(cd shared/llconf && CFLAGS="-fPIC" ./configure --prefix=$(pwd)/install)
(cd shared/llconf && make)
(cd shared/llconf && make install)
(cd shared && ln -s llconf-0.4.6 llconf)

# Build FSEFI
./configure --target-list=i386-softmmu --proj-name=tracecap --prefix=$(pwd)/install --disable-gcc-check
make clean
make
```

3 Configuring a new VM

Generally QEMU is compatible with almost any guest OS that runs on X86 architecture, currently FSEFI is only supporting Linux. We port the linux kernel data structure into FSEFI kernel to extract the OS abstraction like processes. In this way, FSEFI can be aware of the starting of specific application by monitoring the `task_struct` list. This release of FSEFI works with VMs running Ubuntu Linux 9.04 32-bit.

Linux-based VM Image: We have prepared Ubuntu linux 9.04 32-bit VM image for test. You can find the image file named `ubuntu_9.04_32.img` in the installation package. You can clone a snapshot image from the original images using Redirect-on-Write to avoid changing the original image and keep your snapshot image small in size and save time when cloning a new VM. But the snapshot image and the backing image file (where the snapshot image built on) must be kept together. For example if you want to create a snapshot of our provided image file `ubuntu_9.04_32.img` for testing algorithm_A, create a new QCow2 format image file called `ubuntu_9.04_32_algorithm_A.img` using `-b` flag to indicate the backing image. The new image is now a read/write snapshot of the original backing image. Any changes (e.g., adding files/directories and removing files/diretories) to the `ubuntu_9.04_32_algorithm_A.img` will not be reflected in `ubuntu_9.04_32.img`.

A command line to create a snapshot image look like:

```
 % qemu-img create -f qcow2 -b ubuntu_9.04_32.img ubuntu_9.04_32_algorithm_A.img
```

At this point, if you run the backing image and do any changes to the backing image will corrupt all the snapshot images that were built based on the this backing image. Please make sure the image you are going to delete is not the backing image.

You can use qemu-img info to check an image's backing image file:

```
 % qemu-img info ubuntu_9.04_32_algorithm_A.img
 image: ubuntu_9.04_32_algorithm_A.img
 file format: qcow2
 virtual size: 40G (42949672960 bytes)
 disk size: 196K
 cluster_size: 65536
 backing file: ubuntu_9.04_32.img
 Format specific information:
     compat: 1.1
     lazy refcounts: false
```

4 Booting a VM Using FSEFI

After you create your first snapshot image, now you can boot your image using FSEFI. You can create a script named `startvm.sh` under directory of SEFI_CURRENT:

```
#! /bin/sh
# Make sure you have the binary file fsefi
# We assume the snapshot image is located in upper level of SEFI_CURRENT
# Options:
#  -monitor stdio : monitor the VM
#  -m 512         : assign 512MB memory for this VM
#  -k en-us       : keyboard Eng-US
#  -cpu core2duo  : Emulated CPU type
./tracecap/fsefi ../ubuntu_9.04_32_algorithm_A.img -monitor stdio -m 512  -k en-us -cpu core2duo
```

The booted VM is using default user mode network, so you don't have to configure the network for this VM. The advantage is that you can generate many VMs working on different projects but never need to manage the network configuration for the group of VMs. And it donest't require any root/administration privileges. The limitation is that the guest is not directly accessible from the host or the external network. So in your guest VM, if you want to install any packages, run your application or transfering files or codes, you have to access the guest VM using the graphic interface initialized by `startVM.sh` (that is why we use `-monitor stdio` to boot the image).

5 Injecting Faults

Now, assuming that FSEFI is compiled and you have launched your VM image using `startVM.sh`, we now walk through and try out an example of injecting soft errors into you application. In the following sections, we will use different notions to specify the environment that you have to operate on.
[Guest:] to indicate the activity in guest linux session.
[FSEFI:] to indicate the activity in host FSEFI console.
[Host:] to indicate the activity in host operating system.

1. *Generate a simple program in the QEMU image:*

   In the guest Linux session, create a `foo.c` program as follows, and start it: (You can also find this code under `/home/guanxyz/USRC/toy/`)

```
$ cat foo.c
```

```
#include <stdio.h>
#include <stdlib.h>

double mymulfunc(double data){
  int i;
  double output = data;
  for(i=0; i<10; i++)
    output *=1.1;
  return output;
}
double myaddfunc(double data){
  int i;
  double output = data;
  for(i=0;i<10;i++)
    output +=1.1;
  return output;
}
int main(int argc, char * argv[]){
  double data = 1.0;
  data = mymulfunc(data);
  printf("fmul output : %.10f\n", data);
  data = myaddfunc(data);
  printf("fadd output : %.10f\n", data);
  return 0;
}
```

```
$ gcc foo.c -o foo
$ ./foo
fmul output : 2.5937424601
fadd output : 13.5937424601
```

2. *Generate the function symbol table (FST) of target application*

   In the guest Linux session, create a process.py program as follows:

```
#! /usr/bin/env
import re
import os.path
import sys

'''
  This python script will clean the symbol table
  - Qiang Guan, LANL
'''
# Start
if len(sys.argv)<3:
        print "Usage: ./process.py binary_file symbol_table"
        sys.exit()

in_file = sys.argv[1] # Input file is a binary
out_file = sys.argv[2] # Output file will be a symbol table
tmp_file = "all_symbols"
cmd_str = "readelf --syms ./"+in_file+" >> "+tmp_file

if os.system(cmd_str)!=0:
        print "Failed in generating symbol table!"
        sys.exit()

if os.path.isfile(tmp_file)!=True:
        print "Symbol table file %s not exist!" %in_file
        sys.exit()


inputfile = open(tmp_file, "r");
outputfile = open(out_file, "w")
for line in inputfile:
        if re.match("(.*)FUNC(.*)", line): # Keep only FUNC
                if re.match("(.*) 0 FUNC(.*)", line)==None: # Clean empty FUNC
                        outputfile.write(line)
inputfile.close()
outputfile.close()
```

[Guest]:
$ python process.py foo symbol_table_foo

Generated symbol_table_foo:

```
    57: 080484b0     5 FUNC    GLOBAL DEFAULT   13 __libc_csu_fini
    66: 080483c4    62 FUNC    GLOBAL DEFAULT   13 mymulfunc
    69: 080484c0    90 FUNC    GLOBAL DEFAULT   13 __libc_csu_init
    72: 08048402    62 FUNC    GLOBAL DEFAULT   13 myaddfunc
    76: 08048440   102 FUNC    GLOBAL DEFAULT   13 main
```

The offset of memory address for the functions and the size in memory for storing the codes are listed in the second column and third colunm. The last column lists the function name.

3. ***[Guest]** Copy the FST from guest to host.*

   For copying file from guest to host file system under the direcotory of `source-code/tracecap/`, you have to access the guest session and use scp to copy file(s) from guest to host.

   [Guest]:
   $ `scp symbol_table_foo your-id@host-ip:/....../SEFI-source-code/tracecap/`

4. *Configure FSEFI*

   Open the FSEFI configuration file `SEFI-source-code/tracecap/ini/SEFI_conf.ini`

   [Host]:
   $ `vim SEFI-source-code/tracecap/ini/SEFI_conf.ini`

   The directory of symbol_table should be specified under `SEFI_host_sym_table_directory`

```


; Configuration file for SEFI
[general]

; set to 'yes', if 'fadd' type is allowed to inject.
; Default : yes
SEFI_support_fault_fadd = yes

; set to 'yes', if 'fmul' type is allowed to inject
; Default : yes
SEFI_support_fault_fmul = yes

; set to 'yes', if 'cmp' type is allowed to inject
; Default : yes
SEFI_support_fault_cmp = yes

; set to 'yes', if 'xor' type is allowed to inject
; Default : yes
SEFI_support_fault_xor = yes

; set to 'yes', if 'sarl' type is allowed to inject
; Default : yes
SEFI_support_fault_sarl = yes

; set to 'yes', if 'idivl' type is allowed to inject
; Default : yes
SEFI_support_fault_idivl = yes

; set to 'yes', if 'imul' type is allowed to inject
; Default : yes
SEFI_support_fault_imul = yes


```

```
; set to 'yes', if dynamic probability model is enabled
; Default : no
SEFI_support_dynamic_probability = no

; The range of bits that are available to flip (based on 64 bits register)
; Default : 52-62
; Example : 52-62 (inject faults into the exponent area of a 64bits register)
SEFI_support_range_start = 52
SEFI_support_range_end = 62

; integer that less sizeof bit_range
; Default : 1
; Example: if SEF_support_sub_bit_range is set 52-62, then it cannot set larger than 11 bits to inject faults.
;          FSEFI will validate it. And if the number exceeds the sizeof range, number
;          will be set as the sizeof range
SEFI_support_number_of_bits_to_flip = 1

; [double] probability of faults
; Example: - If SEFI_support_fault_probability is set 100, then fault is going to be inject with
;          100 percentages.
;          - 0 means the probability mode is disabled. The number of injections per run depends on the
;          user defined option "-n"
; Defautl : 0
SEFI_support_fault_probability = 0

; [integer] index of the start of fadd opes, where the injector will inject faults
; Default : 0
; Example : 0 (the inject will inject fault from the first Fadd)
SEFI_support_start_index_fadd = 0

; [integer] index of the start of fmul opes, where the injector will inject faults
; Default : 0
; Example : 0 (the inject will inject fault from the first fmul)
SEFI_support_start_index_fmul = 0

; [integer] index of the start of cmp opes, where the injector will inject faults
; Default : 0
; Example : 0 (the inject will inject fault from the first cmp)
SEFI_support_start_index_cmp = 0

; [integer] index of the start of xor opes, where the injector will inject faults
; Default : 0
; Example : 0 (the inject will inject fault from the first xor)
SEFI_support_start_index_xor = 0

; [integer] index of the start of sarl opes, where the injector will inject faults
; Default : 0
; Example : 0 (the inject will inject fault from the first sarl)
SEFI_support_start_index_sarl = 0

; [integer] index of the start of imul opes, where the injector will inject faults
; Default : 0
; Example : 0 (the inject will inject fault from the first imul)
SEFI_support_start_index_imul = 0

; [integer] max dynmic instructions
; defualt : 1 (you can specify it when you have more information from profile)
; Example : 100 (the injection will be randomly chosen from 1st target instruction to 100th
;           if SEFI_support_start_index_fadd = 1 )
SEFI_support_max_dic = 100

[host configuration]

; [string]directory of symbol_table for subroutine injection
; Default: NULL
SEFI_host_sym_table_directory = tracecap/symbol_table_foo


```

5. *Profile target application/function*

- Load the tracecap.so. At the (FSEFI) prompt, say:

```
SEFI(QEMU 0.9.1) monitor - type 'help' for more information
(SEFI)  load_plugin tracecap/tracecap.so
general/trace_only_after_first_taint is enabled.
general/log_external_calls is disabled.
general/write_ops_at_insn_end is disabled.
general/save_state_at_trace_stop is disabled.
tracing/tracing_table_lookup is enabled.
tracing/tracing_tainted_only is disabled.
tracing/tracing_single_thread_only is disabled.
```

```
tracing/tracing_kernel is disabled.
tracing/tracing_kernel_tainted is disabled.
tracing/tracing_kernel_partial is disabled.
network/ignore_dns is disabled.
Enabled: 0x00 Proto: 0x00 Sport: 0 Dport: 0 Src: 0.0.0.0 Dst: 0.0.0.0
Loading plugin options from: /home/guanxyz/FSEFI_CURRENT/tracecap/ini/hook_plugin.ini
Loading plugins from: /home/guanxyz/FSEFI_CURRENT/shared/hooks/hook_plugins
general/SEFI_support_fault_fadd is enabled.
general/SEFI_support_fault_fmul is enabled.
general/SEFI_support_fault_cmp is enabled.
general/SEFI_support_fault_xor is enabled.
general/SEFI_support_fault_sarl is enabled.
general/SEFI_support_fault_idivl is enabled.
general/SEFI_support_fault_imul is enabled.
general/SEFI_support_dynamic_probability is disabled.
SEFI configuration is loaded successfully
SEFI configuration: faulty bits from 52 to 62 ( 1 bit(s)) with probability 0.000000
------ THE EXTENDED PRECISION is ENABLED.
Cannot determine file system type
Cannot determine file system type
Cannot determine file system type
tracecap/tracecap.so is loaded successfully!
(SEFI)  enable_emulation
Emulation is now enabled
```

The warning about `Cannot determine file system type` applies to functionality we won't be using, and can be ignored. `enable_emulation` is required to activate any of instraction tracing hooks; without it, later steps won't see any of the instructions executed.

- Command FSEFI to profile target application & function

```
(SEFI)  profile2func foo mymulfunc
```

- Execute the `foo` in guest session and you will see the printing from FSEFI prompt

[Guest]:

```
$./foo
fmul output : 2.593742
fadd output : 13.593742
```

[FSEFI]:

```
Tracing process :foo PID:2958 func:mymulfunc
module: [foo] 0x08048000 -- 0x08049000
Monitoring :0x80483C4 -- 0x80483FF
Target program 'foo' exits! -1 more runs
```

FSEFI traced process foo with process id 2958 and monitored the memory space `0x080483C4 -- 0x080483FF` as we defined in FST. Whenever Execution Instruction Pointer EIP accessed the memory address within this rang, FSEFI will extract the EIP and verify the EIP pointer. If EIP pointed to a `fmul` operation, FSEFI profiler will add the occurance counter of `fmul` by one

- Stop profiling using FSEFI command `trace_stop`.

[FSEFI]:

```
(SEFI)  trace_stop
```

- Extract profile result from `FSEFI_source/SEFI.log` in Host.

[Host]:

```
[23, Feb,23:42:47] [PROFILE] fadd: 10, fmul: 0, xor:0, cmp:11 sarl:0 idivl:0 imul:0
```

There are 10 `fadd` operations and 11 `cmp` operations used in target application and target function `myaddfunc()`. When configuring the injector, if the start index is set as 0, max of dynamic instructions for injection can be configured as 10 if we target to fadd operation or 11 if we target to cmp operation. The addition of the start index and max number of dynamic instruction count should not exceed the profile result, in this example, which is 10 or 11.

6. *Inject faults into application/function at runtime*

- Setup the injector by commanding in FSEFI prompt:

[FSEFI]:

```
(SEFI)  batch_run -a foo -t fadd -f myaddfunc -m 10 -n 1 -r 1
SEFI: Waiting for process foo to inject 1 'add' fault(s) in function myaddfunc.
Process foo is registered.
```

The manual of `batch_run`:

```
  batch_run [-a <target application>]
            [-t <operation type>]
            [-f <target function>]
            [-m <max number of dynamic instruction count>]
            [-n <number of injections in each run>]
            [-r <number of runs of application>]
```

`-a`
: Define the target application.

`-t`
: Define the type of operations for injections. Currently we suppport `fadd, fmul, xor, cmp, sarl, idivl, imul`. But we are able to add more for customization.

`-f`
: Define the target function. It is optional. If you are injecting faults into application level only, you don't have to set this.

`-m`
: Define the max number of dynamic instruction count. If it is not set, SEFI will load the default value from `SEFI_conf.ini`. If the probablity is not set as 0, this option will be automaically disabled.

`-n`
: Define the number of injections in each run of application. You are able to inject multiple faults by setting proper value here. If the probablity is not set as 0, this option will be automaically disabled.

`-r`
: Define the number of runs of applications. This feature enables the batch mode fault injection campagin. You can write a script to repeat the application in guest and set this option with the exact same number of runs of applications. Let FSEFI to inject fault(s) in each run of the application.

- Execute the `foo` in guest session.

[Guest]:

```
$./foo
fmul output : 2.593742
fadd output : 2754157841571327825859331172157156753408.000000
```

[FSEFI]:

```
 (SEFI)  Process foo starts...
 Tracing process :foo PID:3623 func:myaddfunc
 module: [foo] 0x08048000 -- 0x08049000
 Monitoring :0x8048402 -- 0x8048440
 unmasked Register:40202fff0300c08f, ST0:8
 Eip: 0x0804842C, BitFlipMask: 800000000000000, AfterInjection:2754157841571327825859331172157156753408.000000, BeforeInjection:8.093742 DIC:
 Target program 'foo' exits!
```

Information about the injection is shown in FSEFI prompt as well as in the `SEFI.log` log file for post-process

```
[24,Feb,00:19:30] [INFO] SEFI: Waiting for process foo to inject 1 'add' fault(s) in function myaddfunc
[24,Feb,00:55:22] [INFO] Tracing process :foo PID:3623 func:myaddfunc
[24,Feb,00:55:22] [myaddfunc] FaultyOp:ADD_OP,eip:0x0804842C, Original:8.093742, BitflipMask:800000000000000, AfterInjection:2754157841571
```

The first entry indicates that FSEFI is ready for injection. The second entry indicate the time of finding the application running by SEFI probe. The third entry shows the details about the injection, including type of instruction for injection, EIP while injecting fault, original value before injection, which bit is flipped, value after injection and the dynamic instruction count.

6 Batch Mode

FSEFI allows batch processing for injecting faults into multiple runs of application. This feature especially supports the statistic study of application's vulnerabality. Using option `-r` in FSEFI command `batch_run` to specify the required number of runs of application. All the other configurations provide the same functionalities as the single mode. But with different random seeds, the injection in each trial will be different. For example, bit-flip position, dynamic instruction index, et al. Besides, the user should guarentee that the input data for each trial are same.

The user should also write batch script in Guest VM to repeat the execution of target application and collect the execution results from each run of the application so that the post-process can map the fault injection information to the execution result. You can refer to this script `runfoo.sh`:

```
#! /bin/bash
OUTPUTFILE=result
COUNT=20
rm result
until [ $COUNT -lt 0 ];
do
   ./foo >> result
   let COUNT-=1
done
```

The `batch_run` command should also be specified with same numbers of trials. When the last trail finished it will show "0 more runs".

```
 (SEFI)  batch_run -a foo -t fadd -f myaddfunc -m 10 -n 1 -r 20
 SEFI: Waiting for process foo to inject 1 'add' fault(s) in function myaddfunc.
 Process foo is registered.
 ...
 ...
 Target program 'foo' exits! 0 more runs
```

7 Troubleshooting

This section describes some problems users have experienced when using FSEFI, along with the most common causes of these problems.

1. *SEFI does not begin probing program*
   - Did you remember to `load_plugin` the tracecap.so hooks before running the program?
   - Did you remember to `enable_emulation` before running the program?
   - Did you remember to `reboot_SEFI` before re-running the program? For example, after profiling, you may need to reboot SEFI for the injection.
   - Did you enter correct application name and function name in (`batch_cmd` command)?
2. *Profiling result is not written into SEFI.log*
   - Did you remember to run `trace_stop`?
   - Are you using the *tracecap* plugin? The *tracecap* can be configured to probe only certain types of instructions (for instance, fadd and fmul). Check the plugin settings in the SEFI_conf.ini file.
3. *I cannot find the results of my application*.
   - You have to access the guest linux session and then use `scp` to copy the results from guest to host system. So we strongly suggest that you should write a script for executing mutltipl runs of your applications and keep the results somewhere after each run. You can find an example of a batch script of running `foo`.
4. *Compile warnings about `fastcall`*
   - Are you trying to compile with GCC 3.3? It isn't supported.
5. *XWindow displaying problem*
   - You need to install SDL library to enable the GUI for Guest session (You have to access Guest session to control the injection). If you are accessing the host server remotely, your local machine has to be installed with SDL library. For Mac user, you can install XQuartz (it can be downloaded via http://xquartz.macosforge.org/landing/). Current version is 2.7.7 (tested version with FSEFI).
6. *Mac XQuartz window not working properly*
   - When you use Mac XQuartz to remotely boot VM, if you reattach the external monitor after the laptop has been disconnected and gone in to sleep mode, a new X Window would appear in the lower portion of the screen. Attempts to resize it or drag it into the upper portion fail. It is a bug still unfixed for XQuartz 2.7.7 with OS X Mavericks. We haven't tested it in OS X Yosemite yet.
7. *The profile result makes no sense*
   - Sometimes when you find that your profile results makes no sense, It is probably because you didn't turn off the profiler for your last profile. That may result in the profiler keep counting and produce non-sense number. Please always use trace_stop to turn off the profiler even though that is not the result you want to know. Also if you run reboot_SEFI all the time before you start a new profile, that may keep you away from this problem.
8. *FSEFI can't find a BIOS image or keymap*
   - Either run `make install` to put these in the locations FSEFI is expecting, or give their locations with the -L flag.
9. *How to switch between SEFI/Host and Guest*.
   - For Mac you can use the `[Command]+[tab]` to exit Guest and get back to the Host. For others, you can use [Ctlr]+[Alt] to switch out of the Guest.
10. *More memory than 2024 MB for VM? NO!*
    - Our Guest now only support 32bits linux kernels. So Guest system running 32bits linux kernel can assign a maximium of 2035 MB (acctually less than 2048MB).

8 Acknowledgements

FSEFI is built on TEMU (UC Berkely).